Ejercicios de correspondencias

Ejercicio 1 Escribir una función `void apply_map(list<int> &L, map<int,int> &M, list<int> &ML)` que, dada una lista `L` y una correspondencia `M` retorna por `ML` una lista con los resultados de aplicar `M` a los elementos de `L`. Si algún elemento de `L` no esta en el dominio de `M`, entonces el elemento correspondiente de `ML` no es incluido. Por ejemplo, si `L = (1,2,3,4,5,6,7,1,2,3)` y `M= (1,2),(2,3),(3,4),(4,5),(7,8)`, entonces, después de hacer `apply_map(L,M,ML)`, debe quedar `ML = (2,3,4,5,8,2,3,4)`. Restricciones: No usar estructuras auxiliares. El tiempo de ejecucion del algoritmo debe ser O(n), donde `n` es el numero de elementos en la lista, asumiendo que las operaciones usadaas de correspondencia son O(1). [Tomado en el 1er parcial del 20/4/2006].

Solución

```
1  void apply_map(list<int> &L,
2                 map<int,int> &M,
3                 list<int> &ML) {
4    ML.clear();
5    list<int>::iterator q = L.begin();
6    while (q!=L.end()) {
7      if (M.find(*q)!=M.end()) {
8        ML.insert(ML.end(),M[*q]);
9      }
10     q++;
11   }
12 }
```

Resuelto en el archivo `apply-map.cpp`

Ejercicio 2 Dos corresponencias `M1` y `M2` son inversas una de la otra si tienen el mismo numero de asignaciones y para cada par de asignacion `x->y` en `M1` existe el par `y->x` en `M2`. Escribir una funcion predicado `bool areinverse(map<int,int> &M1,map<int,int> &M2);` que determina si las correspondencias `M1`, `M2` son una la inversa de la otra o no. [Tomado en Primer Parcial 17-SET-2009].

Solución

```
1  //---:---<*>---:---<*>---:---<*>---:---<*>---:---<*>
2  bool areinverse(map<int,int> &M1,map<int,int> &M2) {
3    // Tienen que tener la misma cantidad de asignaciones
4    if (M1.size()!=M2.size()) return false;
5    map<int,int>::iterator q1 = M1.begin(),q2;
6    while (q1!=M1.end()) {
7      int x = q1->first,
8        y = q1->second;
9      // verificar que 'y' tiene un valor asignado
10     // y que es 'x'
11     q2 = M2.find(y);
12     if (q2==M2.end() || q2->second!=x) return false;
13     q1++;
14   }
15   return true;
16 }
```

Resuelto en el archivo `areinverse.cpp`

Ejercicio 3 Escribir una función `void concat_map(map<int,list<int> >& M, list<int>& L);` tal que reemplaza los elementos de 'L' por su imagen en 'M'. Si un elemento de 'L' no es clave de 'M' entonces se asume que su imagen es la lista vacía.Por ejemplo: Si `M=5->(3,2,5),2->(4,1),7->(2,1,1)` y `L=(1,5,7,2,3)` , entonces debe quedar `L=(3,2,5,2,1,1,4,1)`. *Restricciones*: El programa no debe usar contenedores auxiliares. [Tomado en el primer parcial del 2010]

Solución

```
1  void concat_map(map<int, list<int> > &M, list<int> &L)
2  {
3          list<int>::iterator il = L.begin();
4          map<int, list<int> >::iterator im;
5          while (il != L.end())
6          {
7                  il = L.erase(il);
8                  // Si existe en M, se inserta la lista correspondiente
9                  // completa en L, usando la version de insert sobre un rango.
10                 if ((im = M.find(*il)) != M.end())
11                         L.insert(il, im->second.begin(), im->second.end());
12         }
13 }
```

Resuelto en el archivo `concat_map.cpp`

Ejercicio 4 Implemente una función `void cutoff_map(map<int, list<int> > &M,int p,int q);` que elimina todas las claves que NO están en el rango `[p,q)`. En las asignaciones que quedan también debe eliminar los elementos de la lista que no están en el rango. Si la lista queda vacía entonces la asignación debe ser eliminada. Por ejemplo: Si `M=1->(2,3,4), 5->(6,7,8), 8->(4,5), 3->(1,3,7)`, entonces `cutoff_map(M,1,6)` debe dejar `M=1->(2,3,4), 3->(1,3)`. Notar que la clave 5 ha sido eliminada si bien está dentro del rango porque su lista quedaría vacía *Restricciones*: El programa no debe usar contenedores auxiliares. [Tomado en el primer parcial del 2010]

Solución

```
1  void cutoff_map(map<int, list<int> > &M, int p, int q)
2  {
3          map<int, list<int> >::iterator im = M.begin();
4          while (im != M.end())
5          {
6                  if (!(im->first >= p && im->first < q))
7                          M.erase(im ++); // la clave no esta en [p, q)
8                  else
9                  {
10                         list<int> &L = im->second; // Alias
11                         list<int>::iterator il = L.begin();
12                         // Eliminar todos los elementos de L
13                         // que no estan en [p, q)
14                         while (il != L.end())
15                         {
16                                 if (!(*il >= p && *il < q))
17                                         il = L.erase(il);
18                                 else
19                                         ++ il;
20                         }
21                         if (L.empty())
22                                 M.erase(im ++); // La lista quedo vacia
23                         else
24                                 ++ im;
25                 }
26         }
27 }
```

Resuelto en el archivo `cutoff_map.cpp`

Ejercicio 5 Dada una correspondecia `M` y un elemento `x0`, podemos generar una secuencia `(x0,x1,x2,...)` de la forma `x_k+1=M(xk)`. La secuencia se detiene cuando uno de los valores `x_k` generados no pertenece a las claves de `M`. En ese caso la secuencia generada es finita. Por otra parte, puede ocurrir que un elemento de la secuencia se repita, es decir `x_k+m=x_k` con `m>0`. Es obvio que, a partir de alli la secuencia se va a repetir indefinidamente. _Consigna:_ escribir una Escribir una funcion `void cyclic(map<int,int> &M,list<int> &L);` que extrae en `L` todas aquellas claves de `M` que generan una secuencia ciclica infinita. Por ejemplo, si `M=(1,2),(2,5),(3,4),(4,6),(5,2)` entonces `cyclic(M,L)` debe retornar `L=(1,2,5)`. [Tomado en 1er parcial 25-SEP-2008].

Solución

```
1  // Predicado que determina si el elemento x
2  // esta en la lista L
3  int contains(list<int> &L,int x) {
4    list<int>::iterator p = L.begin();
5    while (p != L.end())
6      if (x == *p++) return 1;
7    return 0;
8  }
9
10 //---:---<*>---:---<*>---:---<*>---:---<*>---:---<*>
11 // Verifica si la secuencia generada por x_{k+1} = M(x_k)
12 // es finita o no
13 int is_cyclic(map<int,int> &M, int x) {
14   list<int> L;
15   L.insert(L.end(),x);
16   while (1) {
17     // verifica si x tiene un valor asignado
18     // o no. OJO: no se deben generar asociaciones
19     // espurias.
20     map<int,int>::iterator p = M.find(x);
21     // Si no tiene asignacion entonces es un terminador
22     if (p==M.end()) return 0;
23     // Genera el siquiente elemento de la secuencia
24     x=p->second;
25     // Verifica si ya esta en la lista o no.
26     // Si ya esta entonces es ciclica
27     if (contains(L,x)) return 1;
28     L.insert(L.end(),x);
29   }
30 }
31
32 //---:---<*>---:---<*>---:---<*>---:---<*>---:---<*>
33 // Extrae las claves ciclicas de M recorriendo
34 // las claves y aplicando el predicado 'is_cyclic'.
35 void cyclic(map<int,int> &M, list<int> &L) {
36   map<int,int>::iterator q = M.begin();
37   while (q != M.end()) {
38     if (is_cyclic(M,q->first))
39       L.insert(L.end(),q->first);
40     q++;
41   }
42 }
```

Resuelto en el archivo `cyclic.cpp`

Ejercicio 6 Escribir una función `bool es_biyectiva (map <int,int> &A)` que retorna true si la misma representa una función biyectiva, esto es, si la correspondencia `A` describe una relación *uno a uno*. Por ejemplo, supongamos el conjunto `X = (0,1,2,3,4,5)` y consideremos las correspondencias `A1 = (0,2), (1,5), (2,0), (3,3), (4,4), (5,1)` y `A2 = (0,2), (1,1), (2,0), (3,3), (4,3), (5,1)` . En el primer caso, cada elemento (de 0 a 5) tiene preimagen, por lo que `es_biyectiva (A1)` debe retornar `true`. En cambio, en el segundo caso, los elementos 4 y 5 no tienen preimagen, por lo que `es_biyectiva (A2)` debe retornar `false`.

Solución

```
1  bool es_biyectiva (map <int,int> & A) {
2    map <int,int> B ;
3    map <int,int> :: iterator p, q;
4    int x_dominio ;
5    int y_rango;
6    p = A.begin ();
7    while ( p != A.end () ) {
8      x_dominio = p->first;
9      y_rango   = p->second;
10     q = B.find (y_rango);
11     if ( q == B.end () ) B [y_rango] = x_dominio ;
12     p++;
13   } // end while
14   return B.size() == A.size() ;
15 }
```

Resuelto en el archivo `es_biyectiva.cpp`

Ejercicio 7 Una correspondencia es una _permutacion_ si el conjunto de los elementos del dominio (las claves) es igual al del contradominio (los valores). Consigna: escribir una funcion predicado `bool es_permutacion(map<int,int> &M)` que retorna `true` si `M` es una permutacion y `false` si no lo es. [Tomado en Primer Parcial 27-SET-2007].

Solución

```
1  bool es_permut(map<int,int> &M) {
2    map<int,int> M2;
3    map<int,int>::iterator q = M.begin(), q2;
4    while (q!=M.end()) {
5      M2[q->second] = q->first;
6      q++;
7    }
8
9    if (M.size() != M2.size()) return false;
10   q = M.begin();
11   while (q!=M.end()) {
12     q2 = M2.find(q->first);
13     if (q2==M2.end()) return false;
14     q++;
15   }
16   return true;
17 }
```

Resuelto en el archivo `espermut.cpp`

Ejercicio 8 Implemente una función `void intersect_map(map< string,list<int> > &A, map< string, list<int> > &B,map< string, list<int> > &C)` que a partir de los diccionarios `A` y `B` construye un diccionario `C` de manera que las claves de `C` son la interseccion de las claves de `A` y `B` y para cada clave `k` en `C` la imagen `C[k]` es la interseccion de los valores en `A[k]` y `B[k]`. [Tomado en Primer Parcial 17-SET-2009].

Solución

```
1  // Funcion auxiliar, determina si el elemento 'x' esta
2  // contenido en la lista 'L'
3  bool contains(list<int>&L, int x) {
4    list<int>::iterator q = L.begin();
5    while (q!=L.end()) {
6      if (x==*q) return true;
7      q++;
8    }
9    return false;
10 }
11
12 //---:---<*>---:---<*>---:---<*>---:---<*>---:---<*>
13 void intersect_map(map_t &A, map_t &B, map_t &C) {
14   map_t::iterator qa = A.begin(), qb;
15   // Busca claves que esten en ambos maps
16   while (qa!=A.end()) {
17     qb = B.find(qa->first);
18     if (qb!=B.end()) {
19       // Las claves qa->first y qb->first son las mismas.
20       // la, lb son las listas correspondientes.
21       // lc es la lista que deberia ser la interseccion
22       // en el sentido de conjuntos de la y lc.
23       // Notar que la linea abajo inserta la asignaciona
24       // para qa->first en C y al mismo tiemo obtiene una referencia
25       // al valor (lc)
26       list<int>
27         &la = qa->second,
28         &lb = qb->second,
29         &lc = C[qa->first];
30       // Carga la interseccion de 'la' y 'lb' en 'lc'
31       list<int>::iterator qla = la.begin();
32       while (qla!=la.end()) {
33         // Debe chequear que el elemento de 'la' este en
34         // 'lb', pero que no este ya en 'lc' ya que 'lc'
35         // es un conjunto. Es decir no debe tener elementos
36         // repetidos.
37         if (!contains(lc,*qla) && contains(lb,*qla))
38           lc.insert(lc.end(),*qla);
39         qla++;
40       }
41     }
42     qa++;
43   }
44 }
```

Resuelto en el archivo `intersecmap.cpp`

Ejercicio 9 Dada una correspondencia `M` y asumiendo que es invertible o biunivoca (esto es, todos los valores del contradominio son distintos), la correspondencia 'inversa' `N` es aquella tal que, si `y=M[x]`, entonces `x=N[y]`. Por ejemplo, si `M=(0,1),(1,2),(2,0)`, entonces la inversa es `N=(1,0),(2,1,(0,2))`. Consigna: Escribir una función `bool inverse(map<int,int> &M,map<int,int> &N)` tal que, si `M` es invertible, entonces retorna true y `N` es su inversa. En caso contrario retorna falso y `N` es la correspondencia 'vacia' (sin asignaciones) [Tomado en el 1er parcial del 20/4/2006].

Solución

```
1  bool inverse(map<int,int> &M,map<int,int> &N) {
2    N.clear();
3    map<int,int>::iterator q = M.begin();
4    while (q!=M.end()) {
5      if (N.find(q->second)!=N.end()) {
6        N.clear();
7        return false;
8      }
9      N[q->second] = q->first;
10     q++;
11   }
12   return true;
13 }
```

Resuelto en el archivo `inverse.cpp`

Ejercicio 10 Dada una lista `L` de enteros escribir una función `bool es_constante (list <int> &L)` que retorna true solo si todos sus elementos son iguales. Hacerlo con (i) sólo operaciones del TAD lista y (ii) mediante una correspondencia. Escriba un procedimiento `void imprime (map <int,int> &M);` para imprimir una correspondencia.

Solución

```
1  // version con iteradores sobre lista
2  bool es_constante1 (list <int> & L) {
3    list <int> :: iterator p, q;
4    p = L.begin ();
5    q = p;
6    while (q != L.end () && *p == *q) q++;
7    if (q == L.end ()) return true ;
8    else               return false ;
9  }
10
11 //-------------------------------------------------------------------------
12 // version "larga" con iteradores sobre mapeo
13 bool es_constante2 (list <int> & L) {
14   map <int,int> M ;
15   map <int,int> :: iterator q;
16   list <int> :: iterator p;
17   int x ;
18   p = L.begin ();
19   while ( p != L.end () ) {
20       x = *p;
21       q = M.find (x);
22       if ( q == M.end () ) M [x] = 1 ;
23       p++;
24   } // end while
25   return M.size() == 1 ;
26 }
27
28 //-------------------------------------------------------------------------
29 // version "breve" con iteradores sobre mapeo pero con mas operaciones
30 bool es_constante3 (list <int> & L) {
31   map <int,int> M ;
32   list <int> :: iterator p = L.begin ();
33   while ( p != L.end () ) M [*p++] = 1 ; // puede hacer varias reasign
34   return M.size() == 1 ;
35 }
```

Resuelto en el archivo `lista_cte.cpp`

Ejercicio 11 Escribir las funciones `map2list()` y `list2map()` de acuerdo a las siguientes especificaciones. `void map2list(map<int,int> &M,list<int> &keys,list<int> &vals);` dado un map `M` retorna las listas de claves y valores. `void list2map(list<int> &keys,list<int> &vals,map<int,int> &M);` dadas las listas de claves `(k1,k2,k3...)` y valores `(v1,v2,v3...)` retorna el map `M` con las asignaciones correspondientes `(k1,v1),(k2,v2),(k3,v3),....` (Nota: Si hay *claves repetidas*, solo debe quedar la asignacion correspondiente a la *ultima* clave en la lista. Si hay menos valores que claves, utilizar cero como valor. Si hay mas valores que claves, ignorarlos). [Tomado en Primer Parcial 17-SET-2009].

Solución

```
1  void map2list(map<int,int> &M,
2                list<int> &keys,list<int> &vals) {
3    // limpia las listas
4    keys.clear();
5    vals.clear();
6    map<int,int>::iterator q = M.begin();
7    while (q!=M.end()) {
8      // Para cada asignacion, simplemente carga clave
9      // y valor en la lista correspondiente
10     keys.push_back(q->first);
11     vals.push_back(q->second);
12     q++;
13   }
14 }
15
16 //---:---<*>---:---<*>---:---<*>---:---<*>---:---<*>
17 void list2map(list<int> &keys, list<int> &vals,
18                map<int,int> &M) {
19   M.clear();
20   // 'q' y 'r' iteran sobre posiciones correspondientes
21   // en 'keys' y 'vals'
22   list<int>::iterator q=keys.begin(), r=vals.begin();
23   // Carga la asignacion en M. Si hay claves
24   // repetidas va a quedar la ultima, que es lo que pide
25   // el enunciado.
26   while (q!=keys.end() && r!=vals.end()) M[*q++] = *r++;
27   // Si llegamos aca es porque alguna de los dos listas se acabo.
28   // Si se acabo 'keys' entonces no hay que hacer nada y el
29   // lazo siguiente no se ejecuta ninguna vez. Si se acabo 'vals'
30   // entonces el lazo siguiente asigna a las claves restantes el valor 0
31   while (q!=keys.end()) M[*q++] = 0;
32 }
```

Resuelto en el archivo `map2list.cpp`

Ejercicio 12 Considere una red de n computadoras enumeradas $i = 0, 1, ...(n-1)$ las cuales pueden estar conectadas entre si de diversas maneras. Dicha información de la interconexión se la puede almacenar mediante un vector de n listas cuyos elementos son los enteros $[0, n)$ que designan a cada computadora. Cada sublista V_i enumera las primeras "vecinas" de la computadora i, para $0 \leq i < n$. Por ejemplo, supongamos que el vector de listas $V = [v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9]$, de las 10 computadoras enumeradas entre 0 y 9 es: $V = [(6), (8), (7), (6), (8), (7), (0, 3), (2, 5, 9), (1, 4, 9), (7)]$. Por ejemplo, las primeras vecinas de las computadoras 7 y 8 son son (2,5,9) y (1,4,9), respectivamente. Si hacemos un dibujo de la red resultante (ver programa demo) observaremos que hay dos subredes, a saber, la subred 1 conformada por las computadoras G1 = (0,3,6) y la subred 2 conformada por las computadoras G2 = (1,2,4,5,7,8,9). Ver otro ejemplo en el programa demo. Entonces, dado un vector de listas V que representa una red posiblemente desconexa interesa identificar todas las subredes que pueda contener. Esta tarea se puede resolver construyendo la correspondencia $M = M(X, Y)$ que mapea del dominio $X = [0, n)$ de las computadoras presentes en la red, al rango $Y = [1, g_{max}]$ de las subredes posibles. Un algoritmo que ejecuta esta tarea se resume en el programa demo. Consigna: usando las operaciones de los TAD lista, mapeo y cola, escribir un procedimiento `void map_explora (vector < list <int> > & V, map <int,int> & M)` en el cual, dada una red de n computadoras representada por un vector de listas de enteros `V`, construya dicho mapeo M.

Solución

```
1  void map_explora (vector < list <int> > & V, map <int,int> & M) {
2    queue <int> Q ;
3    map <int,int> :: iterator q;
4    list <int> :: iterator p;
5    int h, k, g, n ;
6    int x_dominio ;
7    int y_rango;
8    g = 1;                          // inicializa contadora de subredes
9    n = V.size();                   // numero de computadoras
10   // revisa cada computadora i=0,1,...,n-1
11   for (int i=0 ; i < n-1 ; i++) {
12     if (M.size() == n) return ; // ya estan todas vistas
13     x_dominio = i ;               // la PC "i" es candidata
14     q = M.find (x_dominio);       // averigua si ya estaba
15     if ( q == M.end () ) {        // entonces no estaba
16       M [x_dominio] = g ;         // y la asigna a la subred "g"
17       Q.push (x_dominio);         // y la encola para luego revisarla
18     } // end if
19     while ( !Q.empty() ) {
20       k = Q.front ();             // descola la PC numeros "k"
21       Q.pop();
22       imprime (Q);
23       // lazo para revisar las primeras vecinas de la PC numero "k"
24       p = V [k].begin ();
25       while ( p != V [k].end() ) {
26         x_dominio = *p++ ;        // la PC numero "*p" es candidata
27         q = M.find (x_dominio); // averigua si ya estaba
28         if (q == M.end()) {       // entonces no estaba
29            M [x_dominio] = g ;  // y la asigna al subred "g"
30            Q.push (x_dominio);  // y la encola para luego revisarla
31         } // end if
32       } // end while
33       imprime (M);
34     } // end while
35     // si quedan computadoras no-vistas, seran de la siguiente subred
36     g++;
37   }
38 } // end i
```

Resuelto en el archivo `map_explora.cpp`

Ejercicio 13 Dadas dos correspondencias `A` y `B`, que asocian enteros con listas ordenada de enteros, escribir una funcion `void merge_map(map<int,list<int>> &A, map<int,list<int>> &B, map<int,list<int>> &C)` que devuelve en `C` una correspondencia que asigna al elemento `x` la fusion ordenada de las dos listas `A[x]` y `B[x]`. Si `x` no es clave de `A`, entonces `C[x]` debe ser `B[x]` y viceversa. Por ejemplo: si `M=(1,2),(2,5),(3,4),(4,6),(5,2)` entonces `cyclic(M,L)` debe dejar `L=(1,2,5)`. [Tomado en 1er parcial 25-SEP-2008].

Solución

```
1  typedef list<int> list_t;
2  typedef list_t::iterator liter;
3  typedef map<int,list<int> > map_t;
4  typedef map_t::iterator miter;
5
6  //---:---<*>---:---<*>---:---<*>---:---<*>---:---<*>
7  // Pone en C la fusion ordenada de 2 listas ordenadas
8  // A y B. Elementos repetidos aparecen tantas veces como
9  // en cada uno de los originales (i.e. preserva el numero
10 // de elementos). OJO L1, L2 y L deben ser
11 // objetos diferentes.
12 void merge(list_t &L1,list_t &L2,list_t &L) {
13   liter
14     q1 = L1.begin(),
15     q2 = L2.begin();
16   // Avanza el menor y pone su valor al final de L
17   while ((q1 != L1.end()) && (q2 != L2.end())) {
18     if (*q1 <= *q2) L.insert(L.end(),*q1++);
19     else L.insert(L.end(),*q2++);
20   }
21   // Inserta las colas de L1 y L2, notar
22   // que no importa el orden ya que a esta altura
23   // alguna de las dos esta vacia
24   while (q1 != L1.end()) L.insert(L.end(),*q1++);
25   while (q2 != L2.end()) L.insert(L.end(),*q2++);
26 }
27
28 //---:---<*>---:---<*>---:---<*>---:---<*>---:---<*>
29
30 // Agrega las asignaciones de A a las de C.  Es decir si A
31 // asigna a x la lista La y C le asigna Lc, entonces despues
32 // de hacer add_map(A,C) en C[x] queda merge(La,Lc)
33 void add_map(map_t &A, map_t&C) {
34   // Recorre las asignaciones de A
35   miter qa = A.begin();
36   while (qa!=A.end()) {
37     // clave de la asignacion
38     int ka = qa->first;
39     // Busca si la clave esta asignada en C
40     miter qc = C.find(ka);
41     // Si la clave no esta asignada en C
42     // crea una asignacion con la lista nula
43     if (qc == C.end()) C[ka] = list_t();
44     // A esta altura C *seguro* tiene una
45     // asignacion para ka
46     qc = C.find(ka);
47     assert(qc != C.end());
48     // hace la fusion de las listas en la lista
49     // temporaria L. OJO, no se puede hacer in-place
50     // es decir merge(qa->second,qc->second,qc->second);
51     // ya que los argumentos de merge deben ser diferentes.
52     list_t L;
53     merge(qa->second,qc->second,L);
54     // Copia el resultado de merge en la asignacion de C
55     qc->second = L;
56     qa++;
57   }
58 }
59
60 //---:---<*>---:---<*>---:---<*>---:---<*>---:---<*>
61 void merge_map(map_t &A, map_t &B, map_t&C) {
62   C.clear();
63   // Simplemente agrega las asignaciones de A y B en C
```

```
64    add_map(A,C);
65    add_map(B,C);
66  }
```

Resuelto en el archivo `mergemap.cpp`